시스템 프로그래밍

Assignment3-2

Class : **A**

Professor : **김태석 교수님**

Student ID : **2022202065**

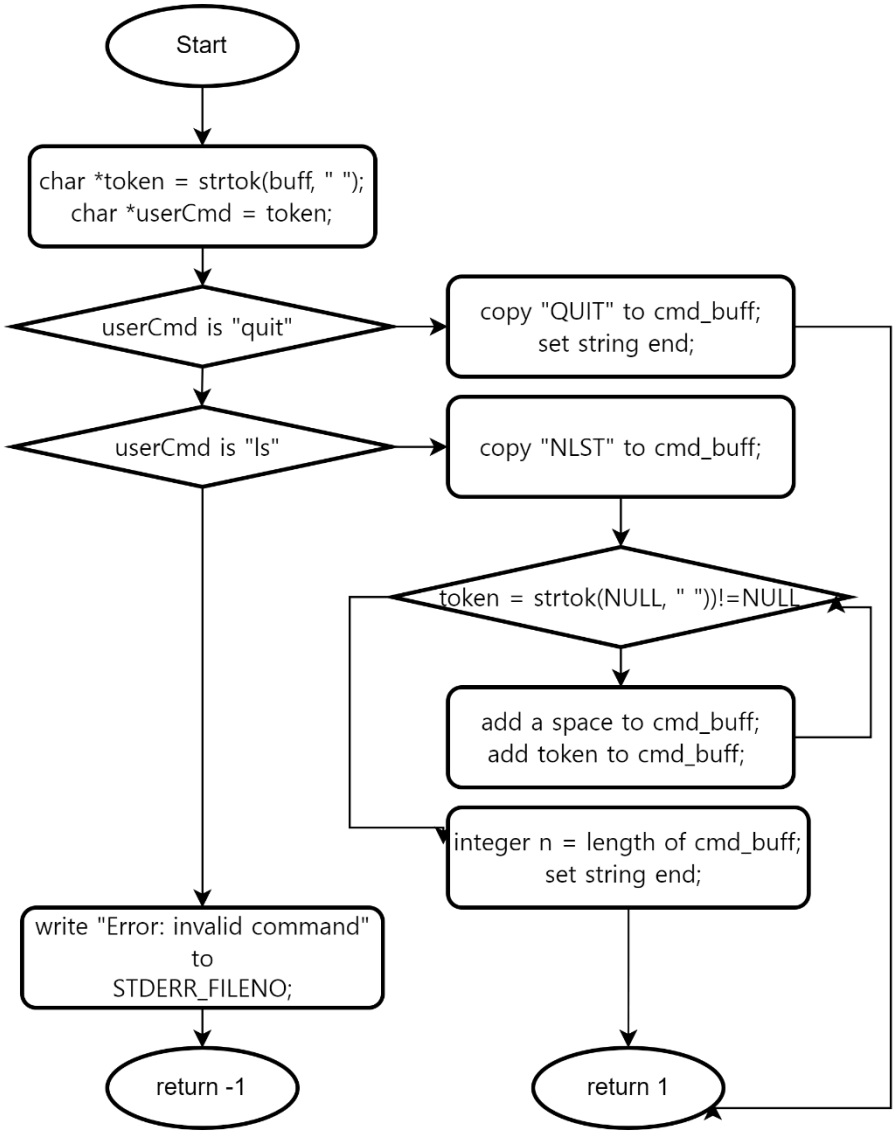
Name : **박나림**

Introduction

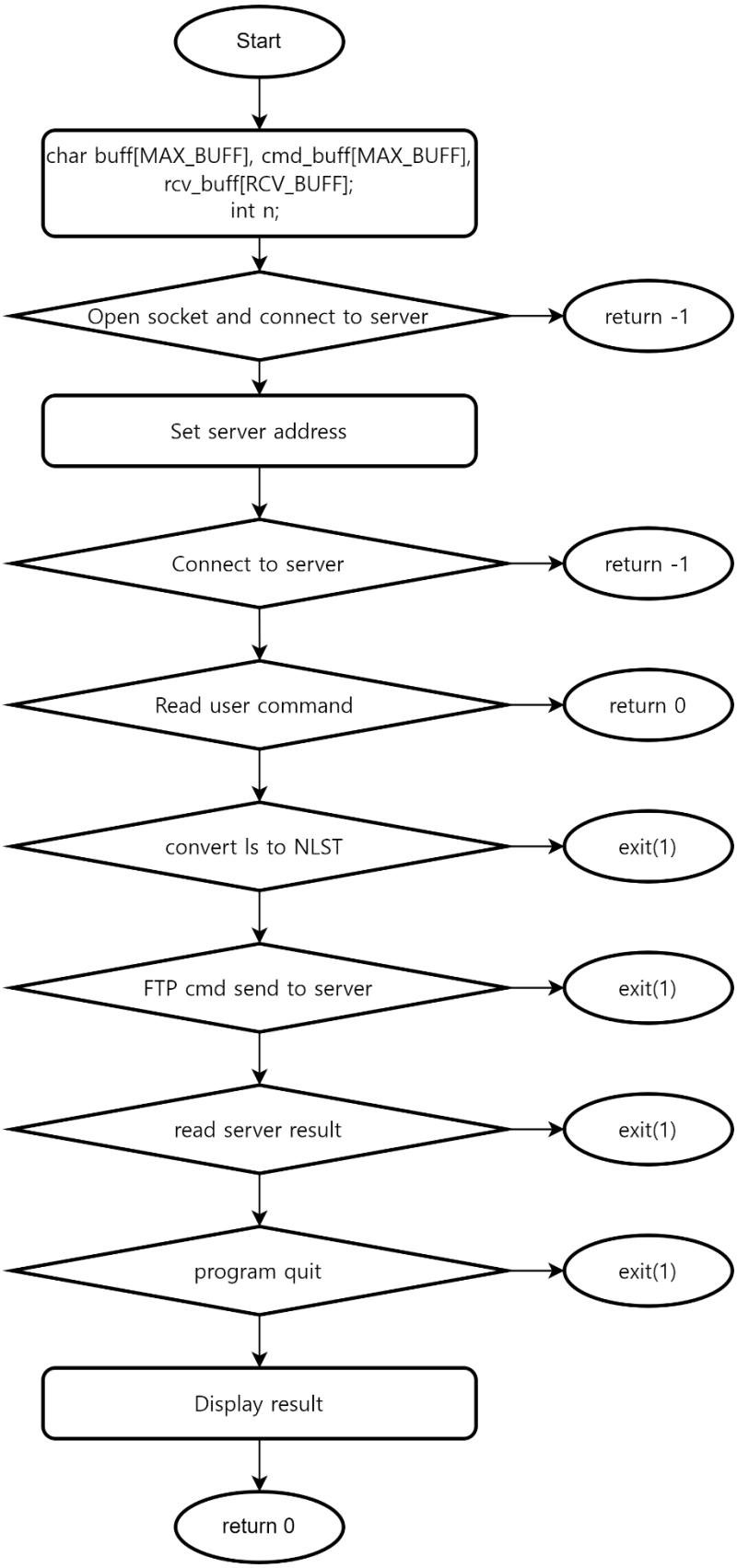
FTP 를 구현하는 프로젝트 중, control connection 과 data connection 을 분리하는 것을 목적으로 하는 과제이다. 클라이언트가 ls 명령어를 입력할 때, control connection 에서 FTP 로 변환된 명령어와 함께 임의로 지정한 새로운 포트 명령어를 같이 서버로 전송한다. 서버에서는 받은 포트번호로 새로 소켓을 생성하여 클라이언트로 접속한다. 이때 이 연결이 data connection 이 되며, 이를 통해 ls 를 수행한 결과를 클라이언트로 보낸다. 과정 별로 성공/실패 메시지는 control connection 을 통해 보내지게 되며, data connection 은 한 번 수행 후 종료되게 한다.

Flow chart

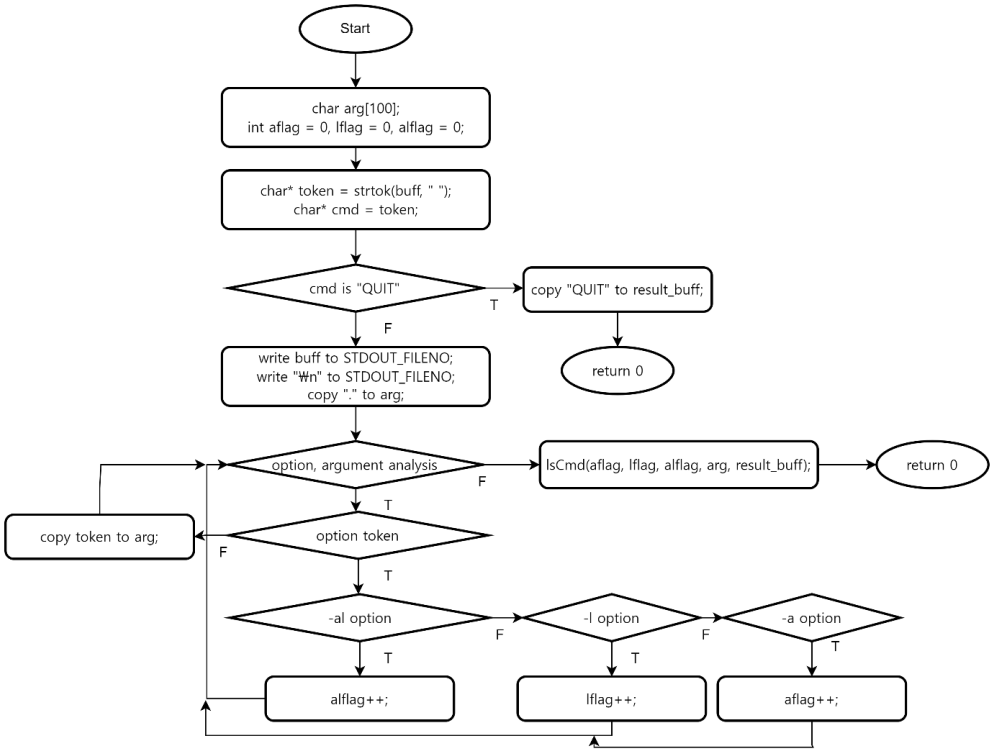
cli.c - conv_cmd()



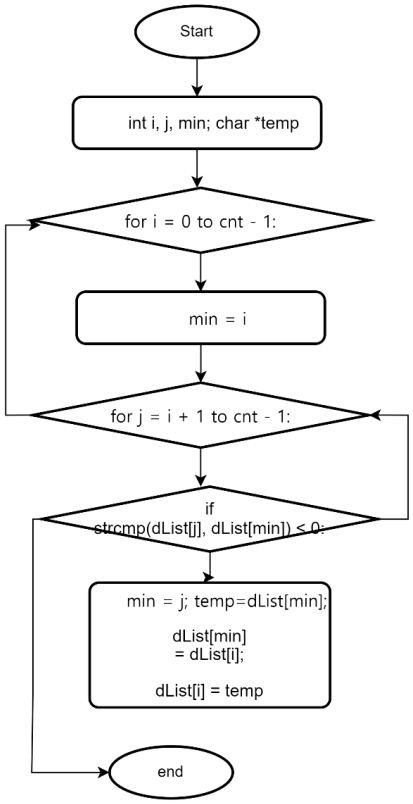
cli.c - main()


Start
char buff[MAX_BUFF], cmd_buff[MAX_BUFF],
rcv_buff[RCV_BUFF];
int n;
Open socket and connect to server
return -1
Set server address
Connect to server
return -1
Read user command
return 0
convert ls to NLST
exit(1)
FTP cmd send to server
exit(1)
read server result
exit(1)
program quit
exit(1)
Display result
return 0

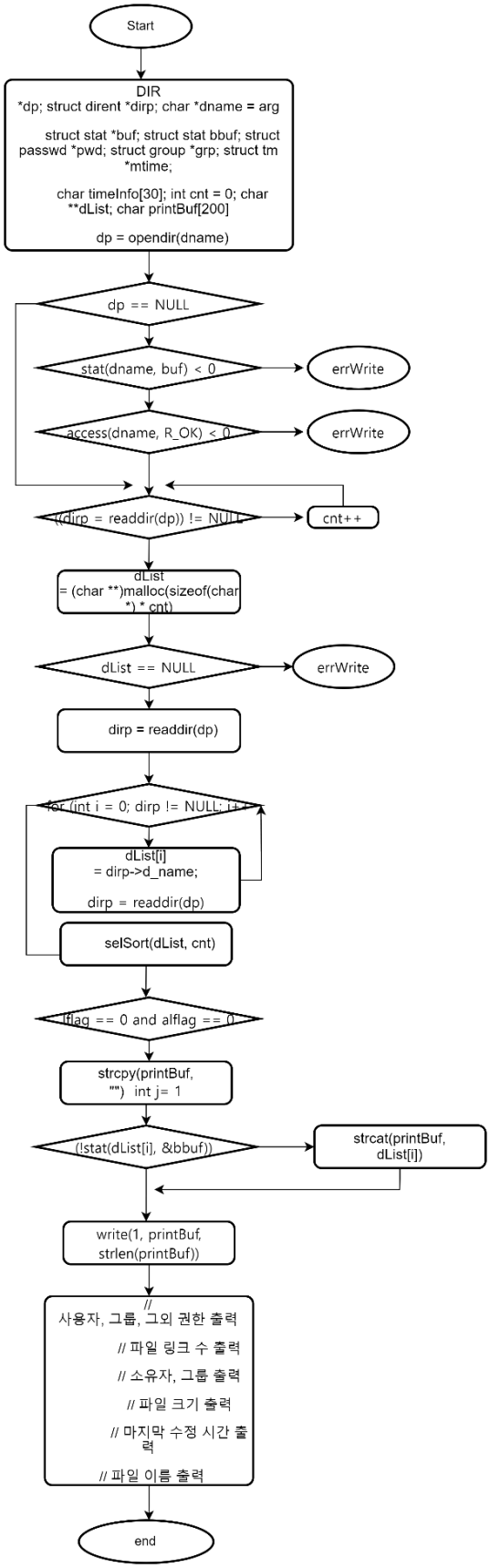
srv.c – cmd_process



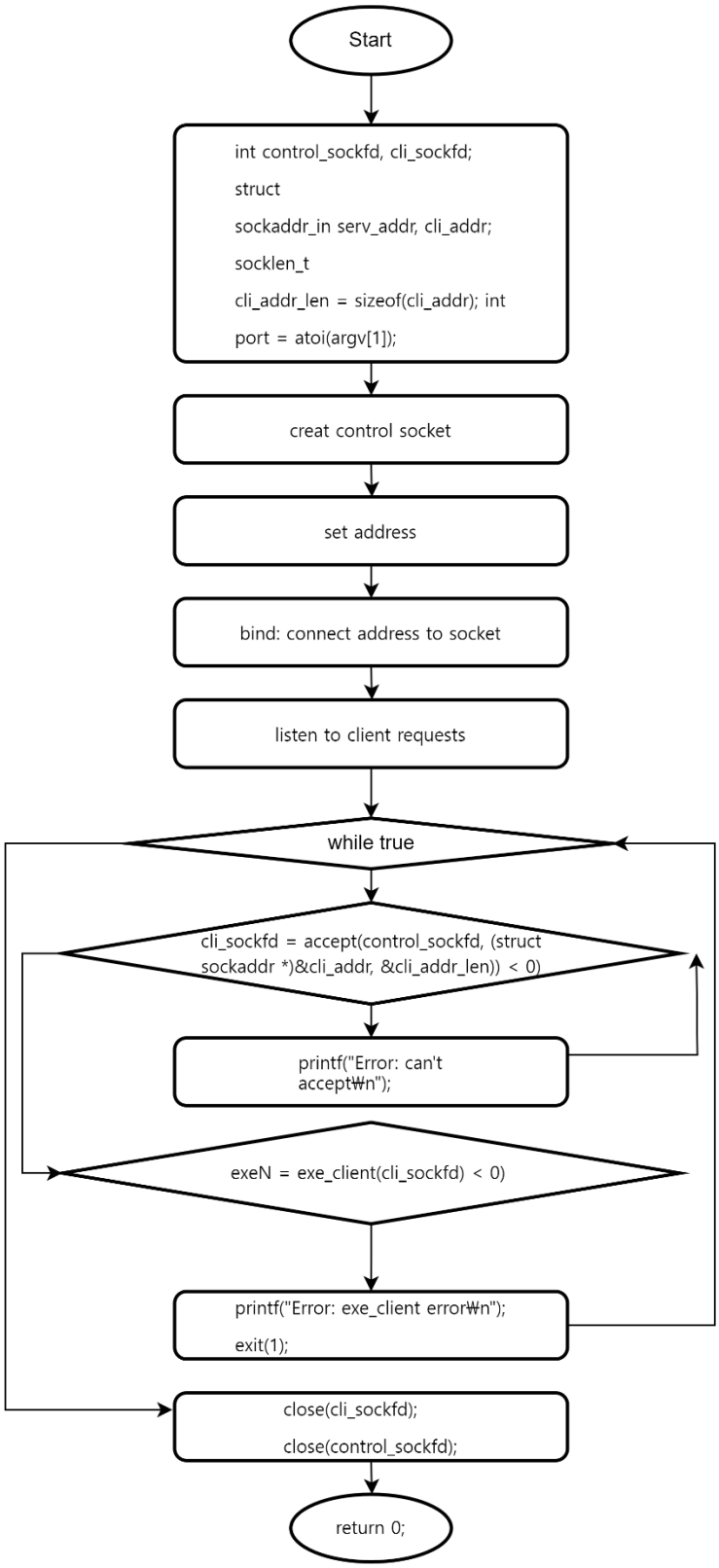
srv.c – selSort



srv.c – lscmd


Start
DIR
*dp; struct dirent *dirp; char *dname = arg
struct stat *buf; struct stat bbuf; struct passwd *pwd; struct group *grp; struct tm *mtime;
char timeInfo[30]; int cnt = 0; char **dList; char printBuf[200]
dp = opendir(dname)
dp == NULL
stat(dname, buf) < 0
errWrite
access(dname, R_OK) < 0
errWrite
((dirp = readdir(dp)) != NULL)
cnt++
dList = (char **)malloc(sizeof(char *) * cnt)
dList == NULL
errWrite
dirp = readdir(dp)
for (int i = 0; dirp != NULL; i++)
dList[i] = dirp->d_name;
dirp = readdir(dp)
selSort(dList, cnt)
lflag == 0 and alflag == 0
strcpy(printBuf, "") int j= 1
(!stat(dList[i], &bbuf))
strcat(printBuf, dList[i])
write(1, printBuf, strlen(printBuf))
//
사용자, 그룹, 그외 권한 출력
// 파일 링크 수 출력
// 소유자, 그룹 출력
// 파일 크기 출력
// 마지막 수정 시간 출력
// 파일 이름 출력
end

srv.c – main()


Start
int control_sockfd, cli_sockfd;
struct
sockaddr_in serv_addr, cli_addr;
socklen_t
cli_addr_len = sizeof(cli_addr); int
port = atoi(argv[1]);
creat control socket
set address
bind: connect address to socket
listen to client requests
while true
cli_sockfd = accept(control_sockfd, (struct sockaddr *)&cli_addr, &cli_addr_len)) < 0)
printf("Error: can't accept₩n");
exeN = exe_client(cli_sockfd) < 0)
printf("Error: exe_client error₩n");
exit(1);
close(cli_sockfd);
close(control_sockfd);
return 0;

Pseudo code

cli.c

```
int conv_cmd(char* buff, char*cmd_buff) {
    char *token = strtok(buff, " ");                    // user command
    char *userCmd = token;
    if userCmd is "quit" then {
        copy "QUIT" to cmd_buff;                        // FTP command
        set string end; }
    else if userCmd is "ls" then {
        copy "NLST" to cmd_buff;                        // FTP command
        while((token = strtok(NULL, " "))!=NULL){ // option, argument parsing
            add a space to cmd_buff;
            add token to cmd_buff; }
        integer n = length of cmd_buff;
        set string end; }
    else {
        write "Error: invalid command" to STDERR_FILENO;
        return -1; }
    return 1
}
```

```
int main(int argc, char *argv[]) {
    int sockfd, data_sockfd, len, n;
    char buff[MAX_BUF], cmd_buff[MAX_BUF], rcv_buff[MAX_BUF];
    struct sockaddr_in serv_addr, data_addr, cli_addr;
    socklen_t cli_len = sizeof(cli_addr);
    char *haddr; //IP address
    haddr = argv[1];
    int PORTNO = atoi(argv[2]);
```

```
Open socket and connect to server
Set server address
If connect to server using sockfd and serv_addr is less than 0 then:
    Output error message "Error: can't connect to server"
    Close sockfd
    Return -1
// Input ls command, convert to NLST
Write "> " to standard output
Clear buff
Clear cmd_buff
len = read from standard input into buff with size MAX_BUF
If len is greater than 0 then:
    Remove newline characters from buff
    If convert ls to NLST(buff, cmd_buff) is less than 0 then:
        Output "Error: invalid command"
        Return -1
    If cmd_buff equals "QUIT" then:
        Output "221 Goodbye"
        Exit program
Send Port command
Write hostport to sockfd
Clear buff
read from sockfd into buff with size MAX_BUF
Write buff to standard output
Write newline to standard output
If first 3 characters of buff are not "200" then:
    Write "Error: not ACK" to standard error
    Return -1
// Data connection
data_sockfd = create socket with domain PF_INET, type SOCK_STREAM, protocol 0
```

```
If data_sockfd is less than 0 then:
    Output "Error: can't create socket"
    Close sockfd
    Return -1
Set new address
If bind data_sockfd with address data_addr is less than 0 then:
    Output error message "Error: can't bind data socket"
    Close data_sockfd
    Close sockfd
    Return -1
If listen on data_sockfd with backlog 1 is less than 0 then:
    Output error message "Error: can't listen"
    Close data_sockfd
    Close sockfd
    Return -1
Declare cli_sockfd as Integer
cli_sockfd = accept connection on data_sockfd with client address cli_addr and length cli_len
If cli_sockfd is less than 0 then:
    Output error message "Error: can't accept data connection"
    Close data_sockfd
    Close sockfd
    Return -1
Write cmd_buff to cli_sockfd
Clear rcv_buff
Clear buff
n = read from cli_sockfd into buff with size MAX_BUF - 1
buff[n] = end of string
Write buff to standard output
n = read from cli_sockfd into rcv_buff with size RCV_BUF - 1
If n is less than 0 then:
```

```
        Write "Error: read() error!!" to standard error
        Exit program
    rcv_buff[n] = end of string
    Display result
    Clear buff
    n = read from cli_sockfd into buff with size MAX_BUF
    buff[n] = end of string
    Write buff to standard output
    Close cli_sockfd
    Close data_sockfd
    Close sockfd
    Return 0
}
```

---

srv.c

```
int cmd_process(string buff, string result_buff)
    char arg[100];                          // ls argument (path)
    int aflag = 0, lflag = 0, alflag = 0;   // ls options (-a, -l, -al)
    char* token = strtok(buff, " "); // command
    char* cmd = token;
    if cmd is "QUIT" then { // quit command
        copy "QUIT" to result_buff;
        return 0; }
    else {                          // ls command execute
        write buff to STDOUT_FILENO;
        write "₩n" to STDOUT_FILENO;
        copy "." to arg;            // set default path to current directory
        while token = strtok(NULL, " ") is not NULL {   // option, argument analysis
            if token[0] is '-' then {                                   // option
                if token[2] is 'l' then increment alflag;               //-al
                else if token[1] is 'l' then increment lflag;           //-l
```

```
                else if token[1] is 'a' then increment aflag; }       //-a
            else      {                                              // no option, argument = path
                copy token to arg; }
        lsCmd(aflag, lflag, alflag, arg, result_buff);                 // ls execute
    return 0
}
```

```
void selSort(array dList[], integer cnt) {
    integer i, j, min; char temp;
    for i = 0 to cnt - 1 { // Selection Sort
        min = I;
        for j = i + 1 to cnt - 1 {
            if strcmp(dList[j], dList[min]) < 0 then {
                min = j;
            }
        }
        temp = dList[min];
        dList[min] = dList[i];
        dList[i] = temp;
    }
```

```
    void lsCmd(int aflag, int lflag, int alflag, char arg, char result_buff) {
        DIR *dp;
        struct dirent *dirp;
        string dname = arg;
        struct stat *buf ;// dList
        struct stat bbuf; // dList[i]
        struct passwd *pwd; // User ID
        struct group *grp; // Group ID
        struct tm *mtime; // Modified time
        char timeInfo[30]; // Time information
        int cnt = 0; // File count
```

```
array dList; // Directory list
char printBuf[200]; // Buffer for print
Open directory;
Counting directory list;
rewinddir(dp); // Reset read position
dList = malloc(sizeof(string) * cnt); // Dynamic allocation of directory list
if dList == NULL then {
    errWrite("empty directory")
}
dirp = readdir(dp);
Read again from the beginning and save to the list;
selSort(dList, cnt); // Sorts the directory list by selection sort
// Execute the ls command by option
if lflag == 0 and alflag == 0 then // No-option or -a
    strcpy(printBuf, ""); // Print buffer initialization
    int j = 1; // Variables for print 4 or 5 per line
    for integer i = 0; i < cnt; i++ { // Print the entire list
        No-option, ignore hidden file continue;
        j++;
        if not stat(dList[i], &bbuf) then { // Distinguish between directory and file
            strcat(printBuf, dList[i]);
            if Directory, add to '/';
            else strcat(printBuf, "₩t");
        }
    }
    strcpy(result_buff, printBuf); // Save ls result
else // -l, -al
    for integer i = 0; i < cnt; i++ {
        -l, ignore hidden files continue;
        if not stat(dList[i], &bbuf) then
```

```
                save directory property;
            }
        }
        set string end;
        close directory;
        free(dList);
    }
```

```
int main(int argc, char *argv[]) {
        int control_sockfd, cli_sockfd;
        struct sockaddr_in serv_addr, cli_addr;
        socklen_t cli_addr_len = sizeof(cli_addr);
        int port = atoi(argv[1]);
        creat control socket
        set address
        bind: connect address to socket
        listen to client requests
        while(1) { //Accept client request
                if ((cli_sockfd = accept(control_sockfd, (struct sockaddr *)&cli_addr, &cli_addr_len)) < 0) :
                        printf("Error: can't accept₩n");
                        continue;
                int exeN; //client execute
        if((exeN = exe_client(cli_sockfd) < 0)) {
                        printf("Error: exe_client error₩n");
                        exit(1);
                }
        close(cli_sockfd);
        }
        close(control_sockfd);
        return 0;
}
```

결과화면

먼저 클라이언트와 서버가 같은 포트 번호 10000 으로 control connection 을 연결한다. 사용자가 클라이언트에서 ls 를 입력하면, 클라이언트는 새로운 임의 포트 번호를 생성하여 포트 명령어를 서버에게 보낸 후 메시지들을 출력한다. 이때 임의 포트로 12345 가 지정되었기 때문에 서버에서도 48, 57(12345)로 출력된다. 그 후 변환된 FTP 문자열을 출력한 뒤, ls 수행 결과를 클라이언트에게 보내며 완료 메시지들을 출력한다.

```
kw2022202065@ubuntu:~/3-2$ ./srv 10000
PORT 127,0,0,1,48,57
200 port command successful
NLST
150 Opening data connection for directory list
226 Result is sent successfully
```

```
kw2022202065@ubuntu:~/3-2$ ./cli 127.0.0.1 10000
> ls
200 Port command successful
150 Opening data connection for directory list
Makefile        cli     cli.c   srv
srv.c
226 Result is sent successfully
```

클라이언트에서 quit 명령어를 입력하면 메시지가 출력되면서 종료된다.

```
kw2022202065@ubuntu:~/3-2$ ./cli 127.0.0.1 10000
> quit
221 Goodbye
```

고찰

이번 프로젝트를 진행하면서 control connection 과 data connection 을 분리하는 과정에서 여러 시행착오가 있었다. control 에서 새로운 port 명령어 문자열을 만들어서 전달하고 서버 프로그램에서 추출하는 것까지는 정상적으로 되었으나, 해당 포트번호로 새로운 data connection 용 소켓을 연결하는 부분에서 오류가 출력되었다. 주로 connection()에서 오류가 발생하여서 ip 와 port 가 제대로 설정이 안된 것 같아 주소를 설정하는 과정을 계속 살펴보았었다. htons()나 inet 함수들을 다른 함수로도 바꿔보았지만 계속 연결에 실패했었다. 그래서 해당 부분쪽에서 디버깅 메시지를 출력하도록 해보았더니 서버와 클라이언트에서 새로운 포트번호가 다르게 출력된 걸 발견하였다. 다시 확인해보니 클라이언트에서 새로운 포트로 명령어 문자열을 생성할 때, port/256, port%256 으로 각각 상위, 하위 8bit 씩 계산했어야 하는데 순서를 반대로 계산해서 나타난 상황이었다. 이에 다시 올바른 순서로 고치니 data connection 이 정상적으로 연결될 수 있었다.

ls 결과를 보낸 뒤에 266 등의 성공 메시지를 보내는 과정에서 버퍼 오류가 발생하였다. 서버에서는 다른 오류 문자들까지 함께 출력되었고 클라이언트에서는 stack 오버플로우가 발생했다는 메시지가 출력되었다. write()로 문자열을 보낼 때 MAX_BUF 로 보낸 것이 오버플로우를 발생시킨 것 같아서 따로 문자열을 저장한 버퍼를 만든 후에 해당 문자열을 strlen()으로 계산해서 보냈더니 정상적으로 출력될 수 있었다.

클라이언트에서 명령어를 반복해서 받으면서 quit 이 들어올 때만 종료되도록 수정하였다가, 원래 ls 명령어도 실행이 안되어서 해당 코드를 완성하지 못하였다. 현재 코드로는 하나의 명령어만 동작되는데, 이 점은 connection 을 분리하는 과정을 다시 공부해봐야할 것 같다.

Reference

강의자료 참고